Iiro Niemi

ANDROID-SOVELLUKSEN TOTEUTUS OPISKELIJAJÄRJESTÖLLE

Tietojenkäsittelyn koulutusohjelma
2017

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

ANDROID-SOVELLUKSEN TOTEUTUS OPISKELIJAJÄRJESTÖLLE

Niemi, Iiro
Satakunnan ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Huhtikuu 2017
Ohjaaja: Nieminen, Hans
Sivumäärä: 35
Liitteitä: 2

Asiasanat: Android, ohjelmointiympäristö, ohjelmistokehittäjät

---

Android Studio on tällä hetkellä ainoa työkalu Android-sovellusten kehittämiseen. Tässä opinnäytetyössä perehdyn Android Studio -ohjelmointiympäristön toiminnallisuuksiin, työkaluihin ja itse ohjelmointiin. Android Studio tarjoaa monipuolisia työkaluja sovelluskehittäjän käyttöön ja onkin hyvä ymmärtää, mikä kaikki on mahdollista nykyaikaisella ohjelmointiympäristöllä.

Ohessa toteutan myös sovelluksen opiskelijajärjestö SAMMAKKOlle. Opinnäytetyössäni tulen käymään läpi, miten RSS-syötettä luetaan ohjelmallisesti ja kuinka ilmoituksia voidaan lähettää sovellukseen.

Android Studio on Googlen julkaisema kehitysympäristö ja siinä on vahvasti riippuvuuksia heidän palveluihinsa. Työssäni en pysty kokonaan sivuuttamaan Googlen rajapintoja, mitä Android Studio hyödyntää osassa toiminnallisuuksiaan.

ANDROID APPLICATION DEVELOPMENT TO THE STUDENT GUILD

Niemi, Iiro
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in information technology
April 2017
Supervisor: Nieminen, Hans
Number of pages: 35
Appendices: 2

Keywords: Android, integrated development environment, software developer

---

Android Studio is the only software for developing Android applications at the moment. In this thesis, i will focus on Android Studio's functionalities, tools and programming itself. Android Studio offers wide variety of tools for software developer and it is important to understand what is possible to achieve with modern development environment.

Besides this thesis im working on application for student guild SAMMAKKO. In this thesis i am going to walk you trough how to read RSS-feed programmatically and how to send notification to your application.

Android Studio is released by Google, so there is a lot of connections to services hosted by them. I am not able to avoid all of these interfaces used by Android Studio.

SISÄLLYS

1 JOHDANTO

Tämän opinnäytetyön aiheena on Android-sovelluksen toteuttaminen Android studio ohjelmointiympäristöllä. Opinnäytetyöni toimeksiantajani on Satakunnan ammattikorkeakoulun opiskelijakunta SAMMAKKO. Tein myös yhteistyötä FeedCowBoy-yrityksen kanssa, joka tarjosi käyttööni Samk-live syötteen omaan sovellukseeni.

Toimeksiantajani SAMMAKKO toivoi mobiilisovellusta, joka kokoaisi monesta eri lähteestä tietoa yhden sovelluksen alle. Tässä opinnäytetyössä toteutan Android-sovelluksen heille ja tarkastelen yleisellä tasolla sovelluksen toteutusta Android studio -ohjelmointiympäristössä sekä Java-ohjelmointikielellä.

Kuten missä tahansa tietojenkäsittelyprojektissa toteutukseen liittyy monia eri vaiheita. Sovelluskehitys Android Studiolla sisältää myös useita eri ulottuvuuksia joista pyrin valitsemaan olennaisimmat sovelluksen toteutusta ajatellen.

Android Studion luonteesta johtuen työni on läheisesti sidoksissa Googlen tarjoamiin lähteisiin.

2 ANDROID-SOVELLUS

Android on maailman suosituin mobiilikäyttöjärjestelmä. Androidia voidaan hyödyntää monissa erilaisissa laitteissa kuten esimerkiksi puhelimissa, kelloissa tai televisioissa (Android.com, 2017).

Android-sovellukset on kirjoitettu Java-ohjelmointikielellä, joka hyödyntää Androidin luokkakirjastoja. Tällä hetkellä käytännössä ainoa työkalu sovelluksen kehittämiseen on Android Studio, koska ADT-lisäosa (Ada Development Tools) Eclipse -kehitysympäristöön on vanhentunut. Eclipse on Eclipse Foundation yhteisön kehittämä vapaan-lähdekoodin ohjelmointiympäristö.

Valmis ohjelma käännetään APK-paketiksi (ohjeet kohdassa 9.2). APK-paketti on tiedostotyyppi, joka paketoi Android-sovellukset yhdeksi tiedostoksi ja näin niitä voidaan levittää Android-käyttöjärjestelmällä toimiviin laitteisiin. APK-paketti voidaan siirtää suoraan puhelimeen ja asentaa näin ilman sovelluskaupan rekisteröintiä. Puhelimen asetuksista tulee sallia tuntemattomat lähteet, ennen kuin käyttöjärjestelmä sallii suoran asennuksen.

3 ANDROID STUDIO

3.1 Yleistä

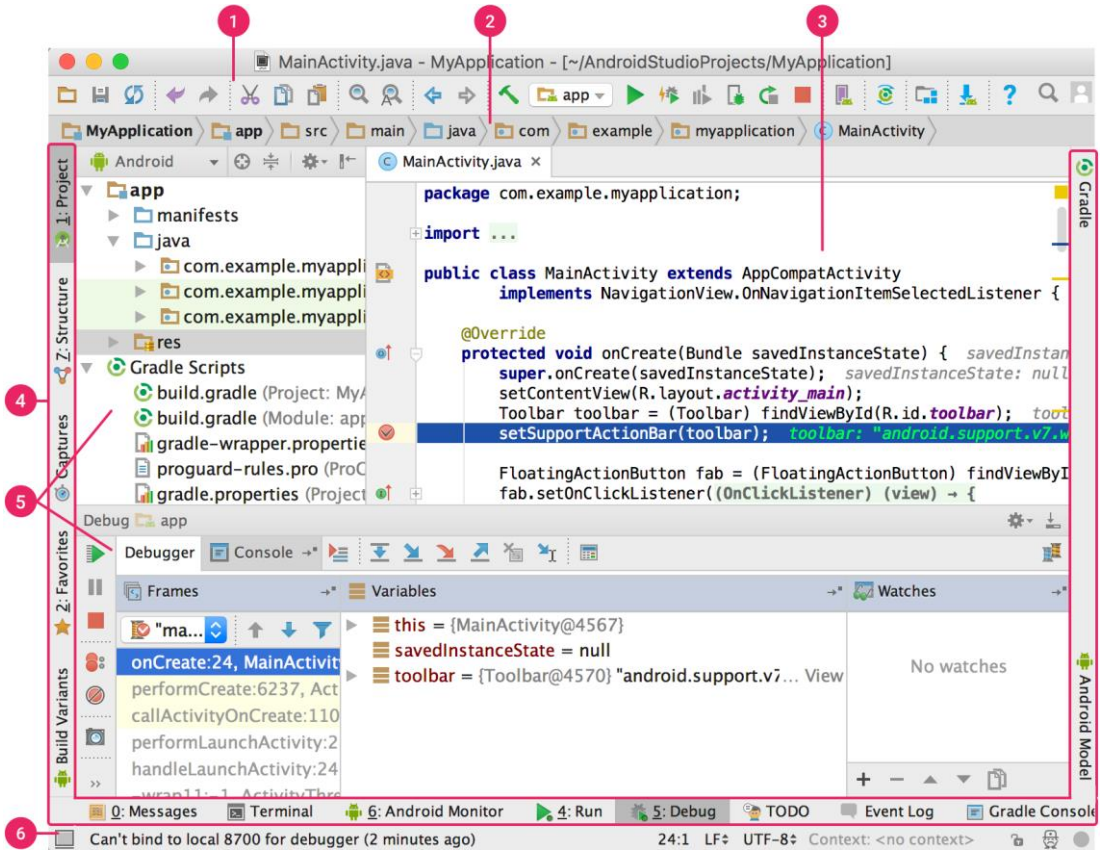
Android Studio on vakiintunut ohjelmointiympäristö Android-sovellusten kehittämiseen. Android Studio perustuu IntelliJ-ohjelmointiympäristöön, jota kehittää JetBrains ohjelmistoyhtiö. IntelliJ on yksi heidän monista tuotteistaan (JetBrains, We are JetBrains 2017).

Android Studioon on lisätty ominaisuuksia sovelluskehityksen mahdollistamiseksi, kuten esimerkiksi puhelinemulaattori, välitön APK:n päivitys ja erilaiset integraatiot eri palveluihin. (Android Studio, 2017a)

3.2 Kehitysympäristö

Tarkastellaan hieman kehitysympäristöä. Android Studiosta löytyy tuttuja rakenteita ja ikkunoita, kuten monissa vastaavissa työkaluissa. Kuvassa 1 on esitelty Android-Studion seuraavat paneelit:

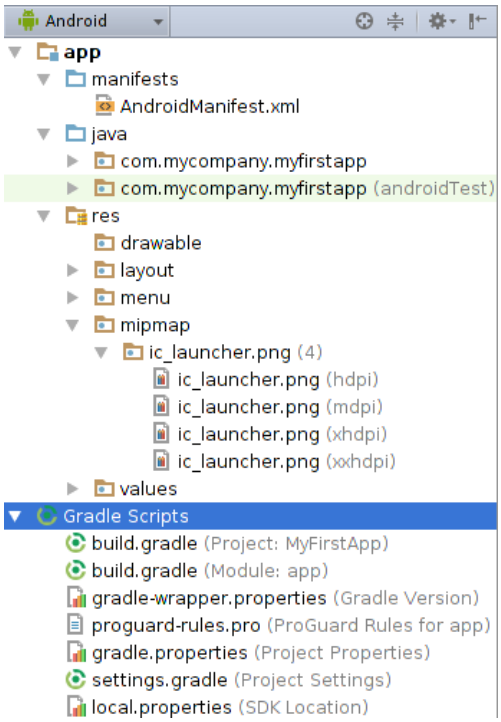
1. Työkalunäkymä sisältää painikkeita sovelluksen käynnistämiseen ja ohjelmakoodin testaamiseen eli debuggaamiseen.
2. Navigaatiopalkki, on kompakti näkymä projektin tiedostoista

3. Editointi-ikkuna jossa tapahtuu ohjelmakoodin muokkaaminen. Xml-malli-pohjia muokatessa käytössä on myös esikatselunäkymä.
4. Työkalukehys kiertää editoria ja tarjoaa erilaisia pikavalintoja.
5. Työkaluikkuna tarjoaa pääsyn tiettyihin tehtäviin, kuten esimerkiksi projektin hallintaa, hakuun ja versionhallintaan.
6. Ilmoituspalkki kertoo projektin ja kehitysympäristön tilan.
7. 

Kuva 1 Android Studion pääikkuna (Android Studio 2017a)

3.3 Kansiorakenne

Oletuksena Android Studio näyttää projektin kansiorakenteisena (Kuva 2), mutta se ei ole suoraan verrannollinen, kuinka tiedostot tallentuvat tiedostojärjestelmään. Osa harvoin käytetyistä tiedostoista ja hakemistoista on piilotettu.



Kuva 2 Kansiorakenne Android Studiossa (Android Studio, 2017b)

Sovelluksen käynnistyksen yhteydessä Android-käyttöjärjestelmä lukee ensimmäiseksi AndroidManifest.xml -tiedoston. AndroidManifest.xml:ssä (Kuva 3) tulee esitellä kaikki sovelluksesta löytyvät komponentit, aktiviteetit ja tarvittavat yhteydet. (Android Studio, 2017c)

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.CALL_PHONE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Kuva 3 Sovelluksen tarvitsemat yhteydet esitellään AndroidManifest.xml:ssä

Kansio java sisältää Java-tiedostot, joihin on ohjelmoitu koko sovelluksen toiminnallisuus. Kansiosta löytyy myös JUnit-testauskehys paikallisen testausrutiinin ohjelmoimiseksi. (Android Studio, 2017c)

Kansio res sisältää kaikki tiedostot, jotka eivät ole suoritettavaa koodia, kuten esimerkiksi XML-tyylipohjat, merkkijonot ja kuvat, kukin sopivaan alikansioon lajiteltuna. (Android Studio, 2017c)

4 KOMPONENTIT

Komponentit ovat Android-sovelluksen rakennuspalikoita. On tärkeää ymmärtää, kuinka erilaiset komponentit keskustelevat keskenään. Toteutin sovellukseni hyödyntäen pääsääntöisesti fragmentteja (tämä termi selitetään tarkemmin luvussa 5.1), eli yhteen aktiviteettiin ladattiin aina uusi fragmentti erilaisilla arvoilla ja näkymillä. Koitan seuraavaksi avata hieman erilaisten komponenttien toimintaa.

4.1 Aktiviteetit

Aktiviteetti kuvastaa sovelluksen yksittäisiä toiminnallisuuksia, jotka ovat selviä kokonaisuuksia sovelluksen sisällä. Esimerkiksi sovellukseni päänäkymä on yksi aktiviteetti (Kuva 4).

Aktiviteetit ovat käynnistettävissä muiden sovellusten kautta, jos sovellus sallii tämän. Käyttöjärjestelmä huolehtii myös käyttäjän liikkumisesta sovelluksessa ja aktivoi aina tarvittavan aktiviteetin näkyville. (Android Studio, 2017c)

Kuva 4 Sovelluksen aktiviteetti värjätty punaisella.

4.2 Palvelut

Palvelut ovat prosesseja, jotka pyörivät sovelluksen taustalla. Ne suorittavat jotain tiettyä tehtävää, esimerkiksi datan siirtoa taustalla tai musiikinsoittoa. Palvelu toimii, vaikka sovellus ei olisi enää aktiivinen. (Android Studio 2017c)

4.3 Sisällöntuottajat

Android-sovellus voi salliessaan tarjota tallentamiaan tietoja tai palveluita muille puhelimen sovelluksille. Tähän ominaisuuteen kulminoituu Android-käyttöjärjestelmän uniikki toiminnallisuus. Esimerkiksi, jos haluat sovelluksesi ottavat valokuvan, sinun ei tarvitse ohjelmoida kokonaista aktiviteettia tätä varten, vaan todennäköisesti puhelimessa on jo toinen sovellus, joka tekee saman asian. Pystyt käynnistämään saman aktiviteetin suoraan omasta sovelluksestasi ja se palauttaa valokuvan ihan kuin se olisi alkuperäinen osa omaa sovellustasi. (Android Studio 2017c)

4.4 Lähetysten vastaanottaminen

Sovelluksesi voi vastaanottaa käyttöjärjestelmän huomioita erilaisista tapahtumista, vaikkei se olisi edes päällä. Näitä huomioita voivat olla sisäiset tai ulkoiset lähetykset, kuten esimerkiksi akun varaustason vähyys (sisäinen), tai että sovelluksen lataus on valmistunut ja data on muiden käytössä (ulkoinen). Kaikista lähetyksistä ei kerrota käyttäjälle. (Android Studio 2017c)

5 NÄKYMIEN KÄSITTELY

5.1 Fragmentit

Fragmentti kuvastaa pientä osaa aktiviteetin käyttöliittymästä. Yksi aktiviteetti voi muodostua useammasta fragmentista, joista jokainen tuo oman näkymän tähän kokonaisuuteen. Fragmenteilla on oma ”elämänkaari” aktiviteetin sisällä ja ne vastaanottavat komentoja aktiviteetin toimiessa, siksi niitä voisi kutsua myös aliaktiviteeteiksi. (Android Studio 2017d)

5.2 Mallipohjat ja niiden näyttäminen

Mallipohja määrittelee käyttöliittymän visuaalisen rakenteen aktiviteetille tai fragmentille. Voit luoda mallipohjia kahdella eri tavalla, määritellä sen suoraan XML-tiedostona, jossa voit hyödyntää näkymän esikatselua ja vedä-pudota -toimintoja.

Toinen keino on luoda komponentti ohjelmallisesti, jolloin pystytään hyödyntämään dynaamista sisältöä paremmin. (Android Studio 2017f)

Staattisen näkymän tuominen esiin tapahtuu suorittamalla onCreate-metodissa setContentView-metodi, johon tulee argumentiksi resurssiviittaus haluttuun mallipohjaan.

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

Kuva 5. Mallipohjan vaihtaminen ajonaikaisesti (Android Studio 2017f).

Esimerkiksi kuvassa 5 suorittuu aktiviteetin tai fragmentin lataamisvaiheessa *onCreate()* -metodi jossa *setContentView()* -metodille annetaan argumenttina viittaus näkymään main_layout.xml.

Metodin *onCreate()* parametrin savedInstanceState sisältöä ei tarvittu tässä esimerkissä, koska näkymä on staattinen eikä siinä ole muuttujia.

5.3 Näytön kierto

Näytön kiertäminen vaakatasoon aiheuttaa sovelluksessa monia toimenpiteitä. Käytännössä käyttöjärjestelmä tuhoaa kaikki aktiviteetit ja fragmentit ja luo ne uudestaan. Tällöin myös paikalliset muuttujat pyyhkiytyvät muistista (esim. lomakkeiden sisältö). Android tarjoaa kuitenkin metodin *onSaveInstanceState(Bundle outState),* jolla sovelluksen tämänhetkisen tilan pystyy tallentamaan muistiin ja palauttamaan takaisin komponenteille tapahtuman jälkeen.

Sovelluksen tilan palauttaminen näytön kierron jälkeen tapahtuu onCreate-metodissa, jossa tutkitaan, onko Bundle-tyyppisen parametrin sisältö muuttunut. Käyttöjärjestelmä kutsuu tätä metodia tapahtuman (näytön kierto) jälkeen, huomaa parametrin tyypitys tallentamisessa ja vastaanottamisessa.

Android-sovelluksissa näytön kierto on oletuksena päällä, tämä toiminnallisuus voidaan estää asettamalla XML-tiedoston aktiviteettiin seuraava attribuutti: *android:screenOrientation="portrait"*. (Tajnar, 2017)

6 SISÄLLÖN HALLINTA

6.1 Viittaaminen sisältöön ohjelmallisesti

Aina jos haluat vaikuttaa ohjelmallisesti sisältöösi, tulee sinun etsiä viittaus kyseiseen resurssiin. Kun Android-ohjelma käännetään, se hakee R-luokkaan kaikki tunnisteet res-kansioosi lataamillesi resursseillesi. (Developers, Accessing Resources 2017)

Android-ohjelmoinnissa resurssiviittauksen syntaksi noudattaa seuraavaa kaavaa:
R.id.objekti → suoraviittaus **näkymän** objektiin id:n perusteella.
R.layout.mallipohjan_nimi → viittaus koko mallipohjaan.

Huomaa että suora viittaus objektiin tapahtuu aina näkymän kautta, eli sinun tulee hakea "kopio" mallipohjasta View-tyyppiseen muuttujaan kuten kuvassa 6.

Kuvassa 6 haen viitteen mallipohjasta oppariohje_layout.xml löytyvään painikkeeseen, jotta voin ohjelmoida tälle painikkeelle toiminnallisuuksia.

```
View view = inflater.inflate(R.layout.oppariohje_layout,container,false);
Button button = (Button) view.findViewById(R.id.tilaa_painike);
```

Kuva 6 Viittaus näkymän sisältöön id:n perusteella

6.2 Merkkijonojen lisääminen

Staattiset merkkijonot, kuten esimerkiksi sovelluksen nimi, painikkeiden tekstit ja tekstimuotoinen sisältö, tallennetaan erilliseen XML-tiedostoon muuttujiksi (Kuva 7). Tällöin sama teksti on aina käytettävissä useammassa paikassa yhden muuttujan kautta.

```
<string name="osoite">"Tiedepuisto 3"</string>
```

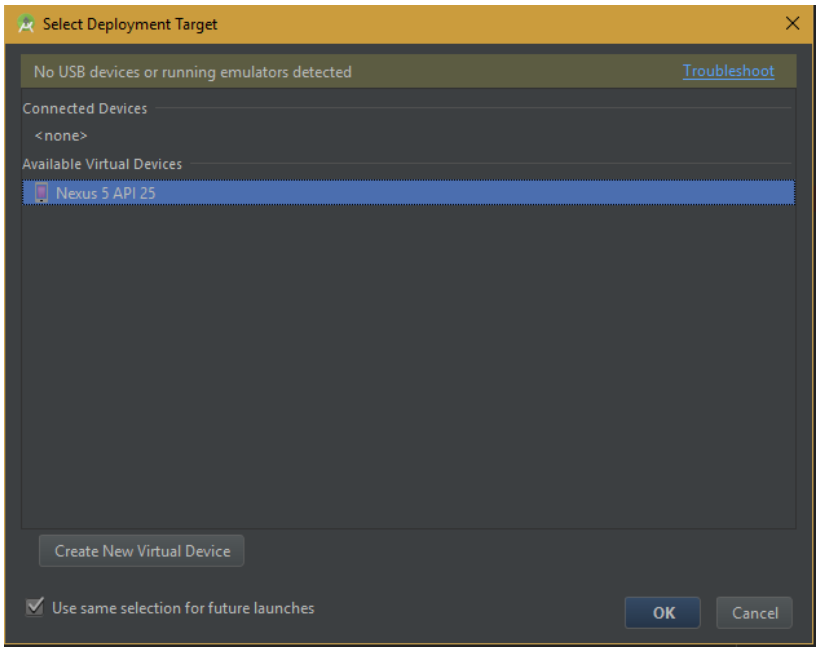
Kuva 7 Merkkijonomuuttuja

6.3 Virtuaaliemulaattorin käyttö

Ennen virtuaaliemulaattorin käyttöä on tärkeää varmistaa, että prosessorisi tukee virtuaalista laitteiston emulointia. Intelin suoritinperheissä tekniikka kulkee nimellä VT-x ja AMD:n suorittimissa nimellä AMD-V. Omalla kohdallani oli käytössä VT-x ja jouduin kytkemään sen päälle BIOSsta. Valikot BIOSissa vaihtelevat emolevykohtaisesti, joten en voi antaa suoraa reittiä tähän asetukseen.

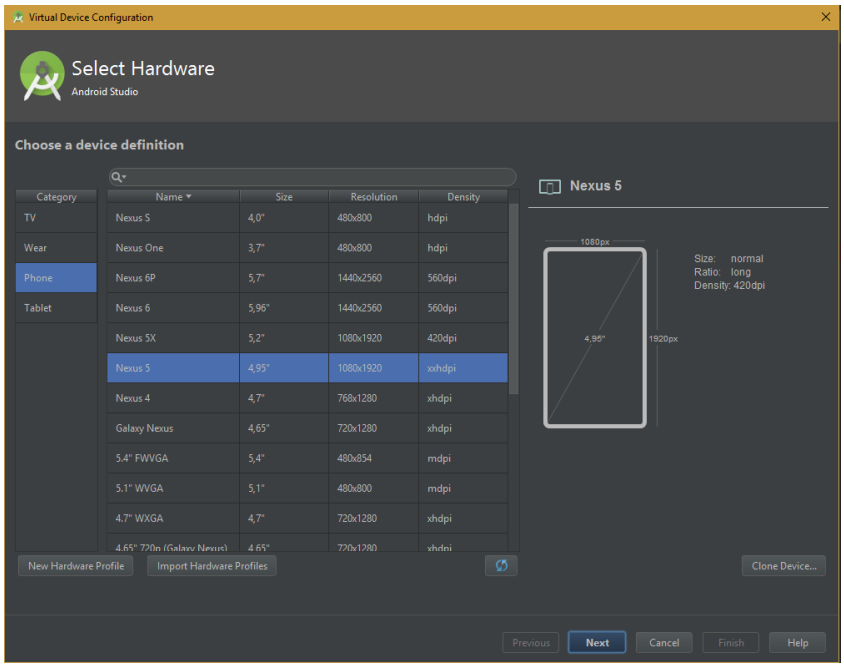
Seuraavaksi opastan miten luodaan uusi virtuaaliemulaattori sovelluskehittämisen helpottamiseksi. Virtuaalinen laite tallentuu myös muille projekteillesi.

Valitse työkalunäkymästä vihreä kolmio tai vaihtoehtoisesti näppäinyhdistelmä ”Shift+F10”. Tämän jälkeen aukeaa ikkuna, joka kysyy, mihin laitteeseen haluat ladata projektin (Kuva 8). Ensimmäistä kertaa projektia emuloitaessa virtuaalilaitteita ei ole yhtään käytettävissä, vaan sellainen tulee luoda.



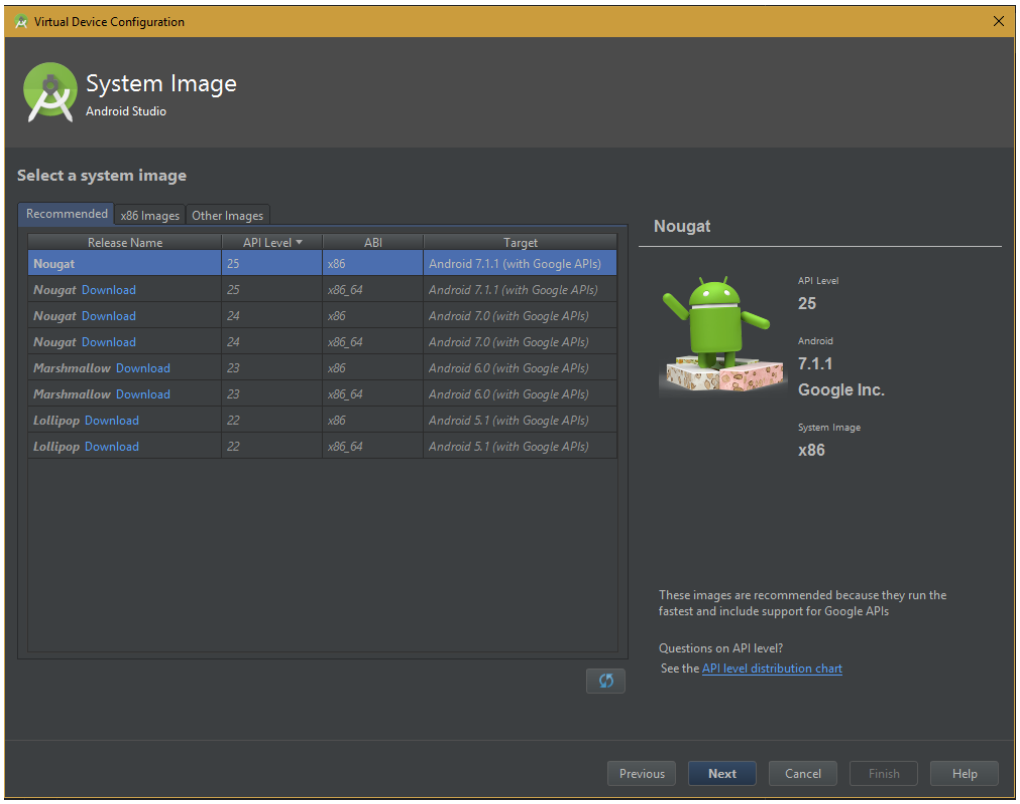
Kuva 8 Valintanäkymä emuloitavalle laitteelle

Valitse ”Create new Virtual Device”, tämän jälkeen sinulta kysytään, millaista laitetta haluat emuloida, tarjolla on yleisimmät Android-laitteiden mallit (Kuva 9). Valittuasi haluamasi laitteen asennusohjelma suosittelee sinulle laitteeseen sopivaa käyttöjärjestelmää (Kuva 10). Käyttöjärjestelmän valinnassa olennaisin seikka on kiinnittää huomiota ”API-level”- arvoon. Tämä arvo kuvastaa, mitä laitteen toimintoja sinulla on käytettävissäsi.



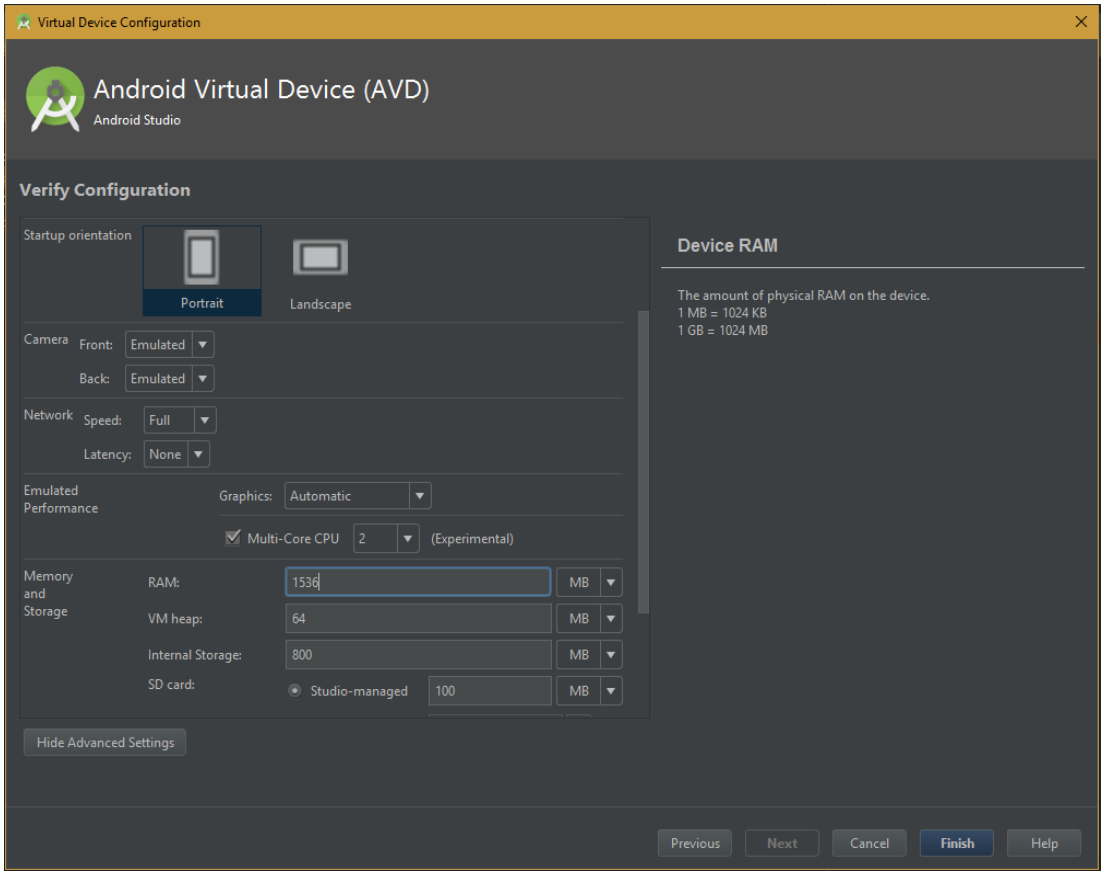
Kuva 9 Valittavissa olevat emuloitavat laitteet

Esimerkiksi sormenjälkitunnistuksen tarjoamat luokkakirjastot tulevat käyttöösi vasta tasolla 23, kun taas XML-tiedoston parsintaan tarvittavat metodit on tuettu jo tasolta yksi lähtien.



Kuva 10 Laitteelle valittavissa olevat käyttöjärjestelmät.

Ennen asennusohjelman loppua voit määrittää emuloitavan laitteen laitteiston määrityksiä kuten sivutusmuistin määrää ja prosessorien lukumäärää (Kuva 11). Asetuksien jälkeen valitse ”Finish”, jolloin Android Studio luo sinulle uuden virtuaalilaitteen, jota voit hyödyntää sovelluksen emuloinnissa.



Kuva 11 Emuloitavan laitteiston määrityksiä.

7 ASYNKRONINEN RSS-SYÖTTEEN LUKEMINEN

7.1 Asynkronisuuden toteuttaminen

Toteuttaessani RSS-syötteen lukua sovellukseeni törmäsin seuraavaan virheilmoitukseen "*NetworkOnMainThreadException*". Tämä virheilmoitus tarkoittaa, että sovellukseni yrittää ajonaikaisesti suoraan internettiin, mikä voisi johtaa käyttöliittymän "jäätymiseen", jos ladattavaa sisältöä joudutaan odottamaan. Android Studion kääntäjä pakottaa Internet-yhteyttä tarvitsevat sovellukset asynkronisiksi (Develop, Exceptions 2017).

Asynkronisuus tarkoittaa ei reaaliaikaista toimintaa, missä ohjelmakoodin valmistuminen odottaa määrittelemättömän ajan saapuvaa vastausta.

Yksi ratkaisu asynkronisuuden toteuttamiseen on hyödyntää rajapintoja (interface). (stackoverflow.com HelmiB 2012.)

Sovelluksessani luokka, joka listaa rss-muotoiset opiskelijaedut (edutFragment), toteuttaa myös rajapinnan AsyncResponse (Kuva 13) ja kutsuu samalla LoadRssFeed luokkaa missä tapahtuu asynkroninen tiedonhaku internetistä (Kuva 12).

```
public class edutFragment extends ListFragment implements AsyncResponse{
    LoadRssFeed asyncTask = new LoadRssFeed();
```

Kuva 12 Fragmentti kutsuu asynkronista luokkaa LoadRssFeed

AsyncResponse-rajapinnan toteuttava luokka (tässä tapauksessa edutFragment luokka) pakotetaan myös toteuttamaan *processFinish()* -metodi missä vastaus palautuu parametrin output kautta.(Kuva 13).

```
package com.samk.iiro.sammakko;

import java.util.ArrayList;

/**
 * Created by Iiro on 12.4.2017.
 */

public interface AsyncResponse {
    void processFinish(ArrayList output);
}
```

Kuva 13 edutFragment.java toteuttaa tämän rajapinnan joka toimii välittäjänä.

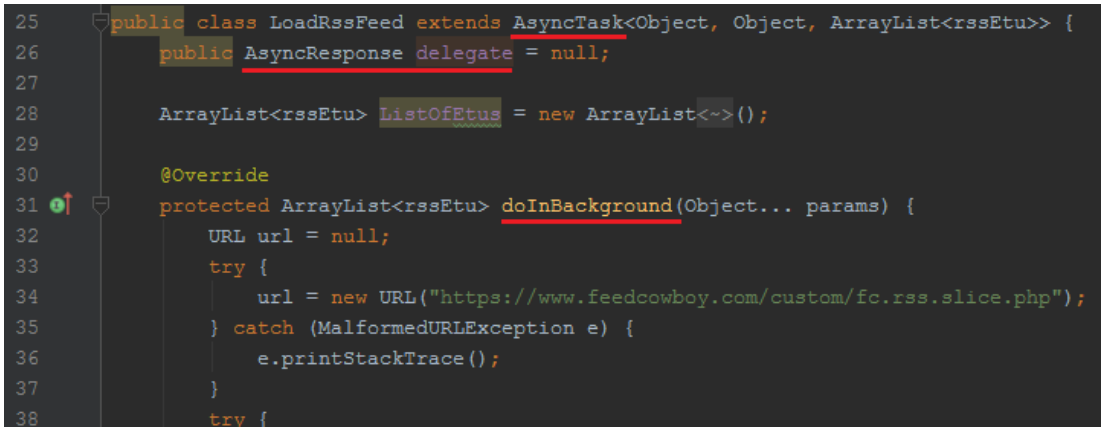
Kun tämä fragmentti (edutFragment.java) joskus myöhemmin käynnistetään, fragmentin luomisvaiheessa ajetaan myös LoadRssFeedin muuttujan (asyncTask) execute -metodi mikä aiheuttaa asynkronisen käsittelyn (kuva 14). Huomaa suora viittaus luokan LoadRssFeed muuttujaan rajapinnan läpi ”*asyncTask.delegate=this;*”.

```
@Override
public void onCreate(Bundle savedInstanceState){
    super.onCreate(savedInstanceState);

    asyncTask.delegate = this;
    asyncTask.execute();
}
```

Kuva 14 edutFragmentin *onCreate()* -metodi; Delegaatin asettaminen Fragmentin käynnistyksen yhteydessä.

Kuorman käsittely tapahtuu luokassa LoadRssFeed, joka laajentaa AsyncTask kirjastoa. Kirjaston käyttäminen pakottaa luokan ylikirjoittamaan vähintään yhden metodin; *doInBackGround()*, mutta on suositeltavaa että ylikirjoitat myös *onPostExecute()* -metodin jolla saat kiinni vastauksesta kun se valmistuu (Developers, AsyncTask 2017).

Kuvassa 15 rivillä 26 alustetaan rajapinnan toteuttava muuttuja, jotta voidaan palauttaa vastaus tämän viitteen kautta *onPostExecute()* -metodissa (katso Kuva 16).



Kuva 15 Asynkroninen käsittely

Kuvasta 16 näkyy, kuinka luokan LoadRssFeed *onPostExecute()* -metodi palauttaa vastauksen rajapinnan toteuttavan muuttujan (delegate) kautta *processFinish()* -metodille edutFragment.java tiedostoon (Kuva 17), koska nämä molemmat tiedostot toteuttavat saman rajapinnan (LoadRssFeed tosin vain delegaatin muodossa).

```
@Override
protected void onPostExecute(ArrayList<rssEtu> result) {
    delegate.processFinish(result);
}
```

Kuva 16 LoadRssFeed luokan *onPostExecute()* metodi lähettää vastauksen rajapinnalle.

Listaksi parsittu data vastaanotetaan luokassa edutFragment ja ladataan toiseen listamuuttujaan näkymäksi muuttamista varten (Kuva 17 rivi 42).

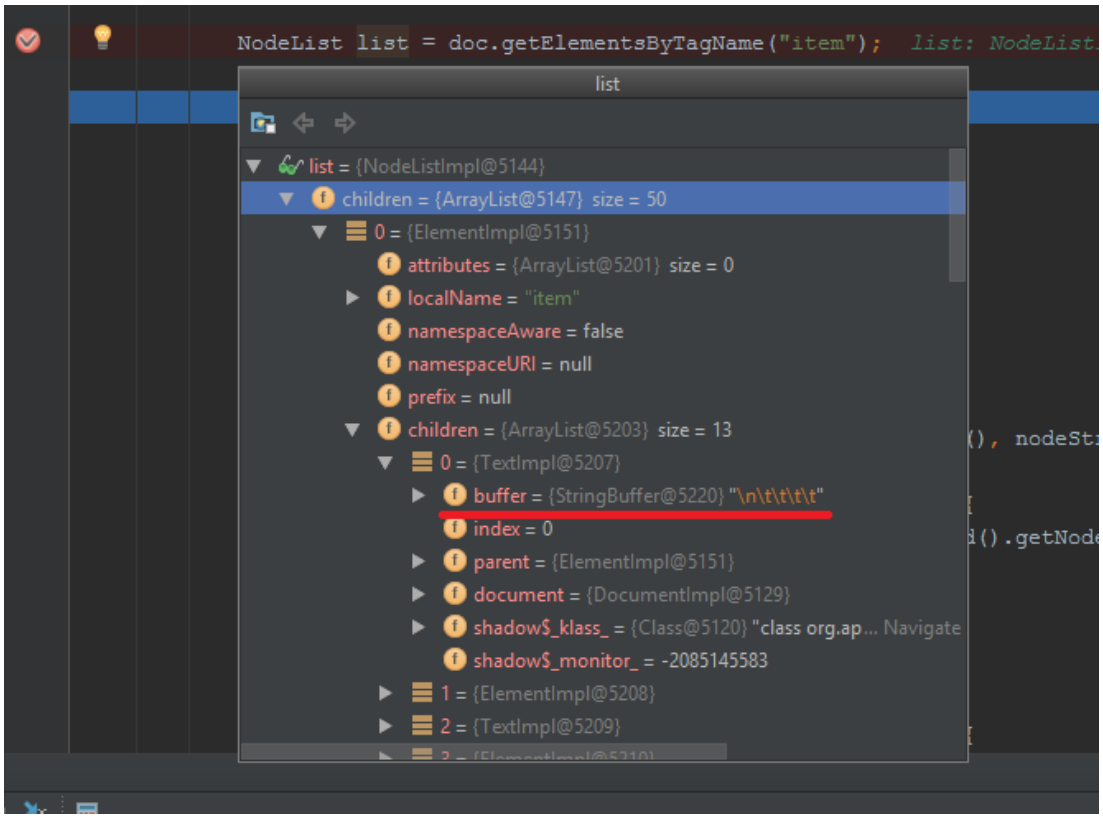
```
39  public class edutFragment extends ListFragment implements AsyncResponse{
40      LoadRssFeed asyncTask = new LoadRssFeed();
41
42      ArrayList ReadyList = new ArrayList();
43
44      @Override
45      public void processFinish(ArrayList output) {
46          ReadyList.addAll(output);
47      }
48
```

Kuva 17 edutFragment luokan metodi *processFinish()* vastaanottaa vastauksen LoadRssFeed luokalta.

7.2 RSS-syötteen lukeminen

Sovellukseni lukee opiskelijaetuja FeedCowboy Oy:n palvelusta RSS-muotoisena osoitteesta https://www.feedcowboy.com/custom/fc.rss.slice.php . Selain ymmärtää kyseisen syötteen RSS-syötteeksi, mutta törmäsin ongelmiin, kun yritin ohjelmallisesti parsia kyseistä syötettä omassa sovelluksessani.

Suurimman ongelman muodostivat “tyhjät välit” elementtien välissä, eli syöte ei ollut täysin validia, kuten kuvassa 18 näkyy.



Kuva 18 Syötteestä löytyi ylimääräisiä erikoismerkkejä.

Toisen ongelman muodostivat syötteessä olevat kuvien URL-osoitteet ja niiden lataaminen erikseen, mutta siihen löytyi kätevä ratkaisu Picasso-lisäosalla.
(Square inc, Picasso 2017)

Hyödynsin RSS-syötteen lukemisessa DocumentBuilderFactory -luokkaa josta erikoistetaan yksittäinen DocumentBuilder olio (Kuva 19). DocumentBuilderFactory vaaditaan, jotta voidaan antaa tarvittavia parametreja syötteen lukijalle, ennen kuin syötteen lukeminen aloitetaan.

```
DocumentBuilderFactory DBF = DocumentBuilderFactory.newInstance();
DBF.setIgnoringElementContentWhitespace(true);
DocumentBuilder builder = DBF.newDocumentBuilder();



Document doc;
doc = builder.parse(new InputSource(url.openStream()));
doc.getDocumentElement().normalize();

NodeList list = doc.getElementsByTagName("item");
```

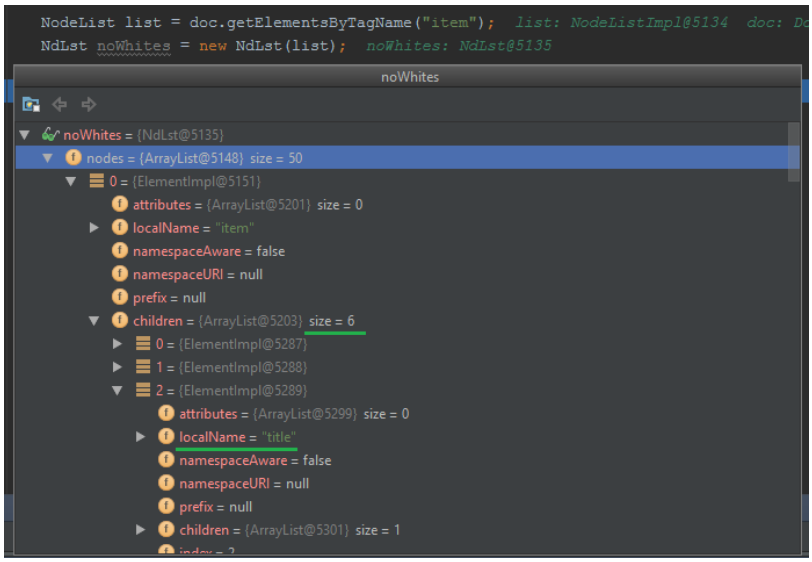
Kuva 19 DocumentBuilder -olion luominen

Keskustelufoorumeilla monet kehittäjät neuvoivat antamaan DocumentBuilderFactory:lle seuraavan määrityksen *.setIgnoringElementContentWhitespace(true);* ennen dokumenttilukija -olion luomista. Tämä määritys ei kuitenkaan tuottanut haluttua lopputulosta, vaan minun tuli parsia tyhjät merkit “käsin” syötteen seasta. Kuvassa 20 hyödynnän “Regular expression”- ilmaisua parsiakseni tyhjät elementit listalta.

```
public static Node ParseWhiteSpaces(Node n) {
    NodeList dirtNodes = (NodeList)n.getChildNodes();

    for (int i = 0; i < dirtNodes.getLength(); i++) {
        Node CN = (Node)dirtNodes.item(i);

            if(CN.getTextContent().contains("\n\t\t\t\t") || CN.getTextContent().contains("\n\t\t\t")){
                n.removeChild(CN);
            } else { }
    }

    return n;
}
```

Kuva 20 Tyhjien merkkijonojen poistaminen syötteen seasta

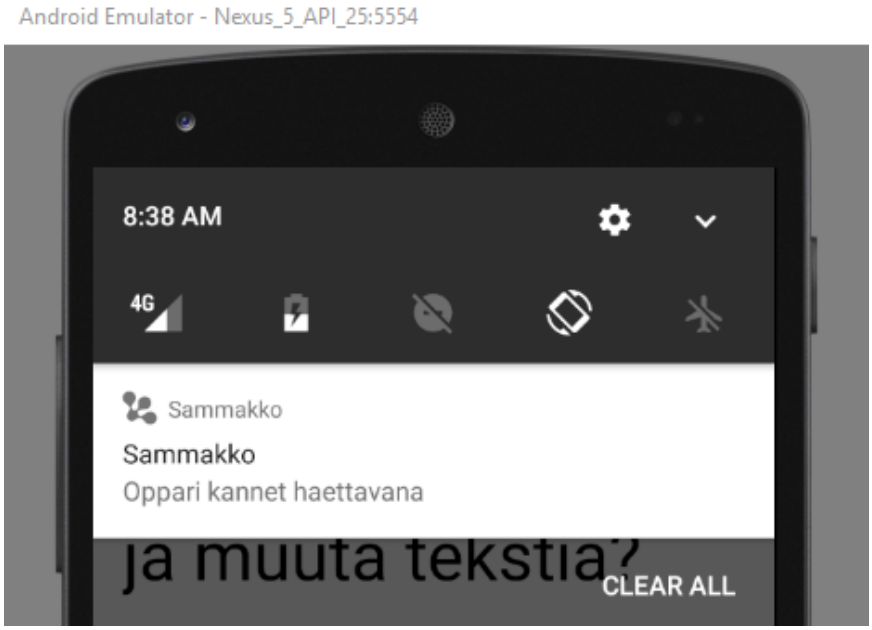
Puhdistettu lista näkyy kuvassa 21. Yhden item-elementin lapsien lukumäärä täsmää nyt syötteessä näkyvään rakenteeseen ja se voidaan nyt esittää käyttäjälle.



Kuva 21 Puhdistettu syöte

8 FIREBASE JA ILMOITUKSET

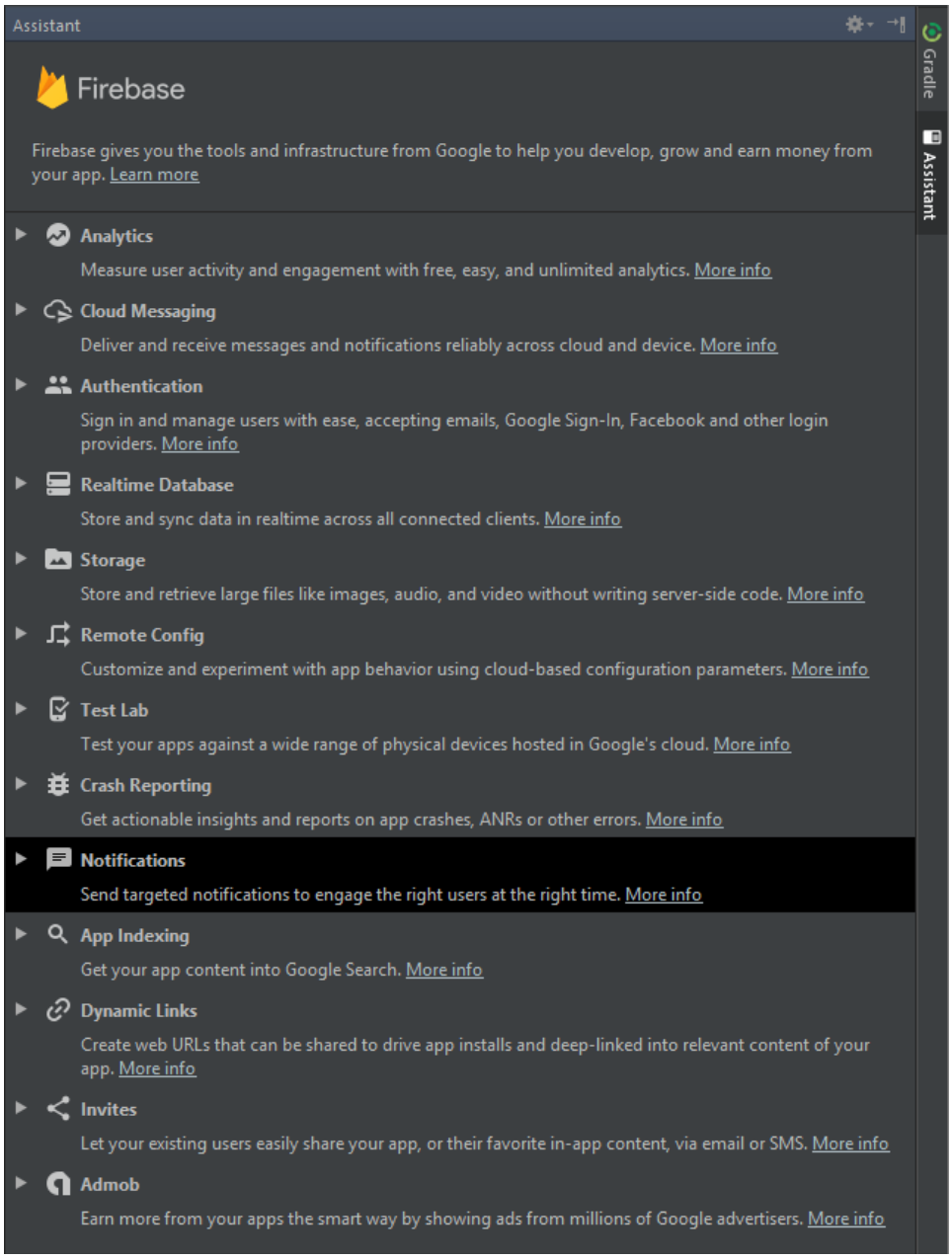
Toteutin sovellukseeni Firebase-integraation, jotta asiakkaan on helpompi lähettää ilmoituksia käyttäjille (kuva 22). Firebase on mobiili- ja websovelluskehitysalusta, joka tarjoaa online-työkaluja kehittäjille. Sovelluksen julkaisun jälkeen tein asiakkaalle oman profiilin Firebaseen ja annoin tälle oikeudet sovellukseeni. Opastin myös, miten ilmoituksia lähetään käyttäen konsolia.



Kuva 22 Testi-ilmoitus

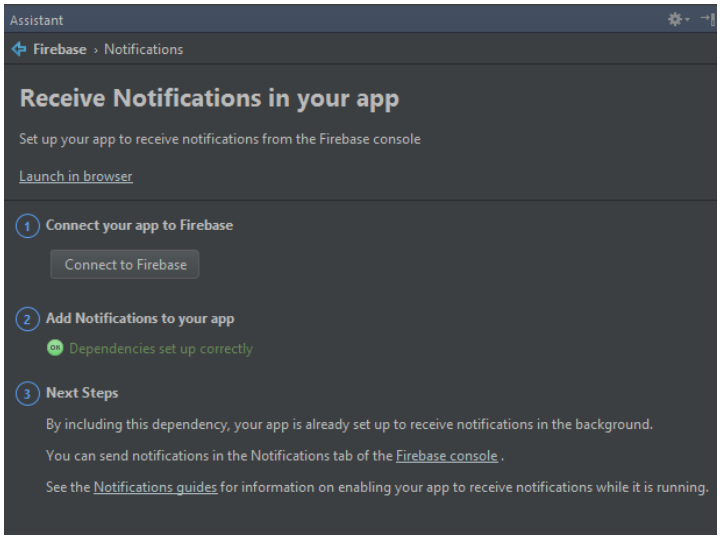
8.1 Ilmoituksen määrittely ja niiden lähettäminen

Android Studiosta löytyy ohjattu Firebasen käyttöönotto**.** Työkalussa voidaan avata ikkuna, jossa on kaikki Firebasen tarjoamat palvelut (kuva 23).



Kuva 23 Firebase:n tarjoamia palveluja, jotka voidaan lisätä suoraan sovellukseen.

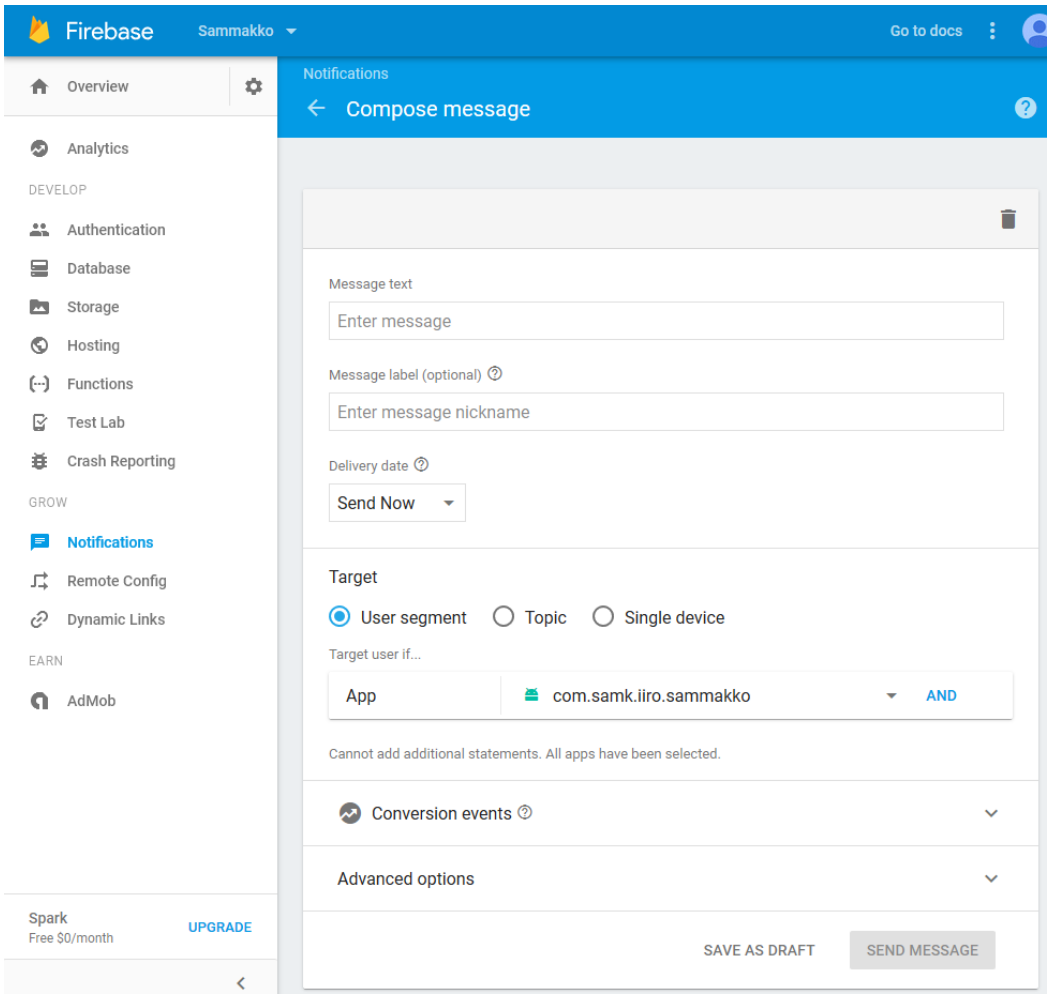
Ohjattu asennus tekee tarvittavia muutoksia suoraan projektisi käännöstiedostoihin (Gradle scripts). Sinun tulee ainoastaan kirjautua sisään kehittäjäprofiililla tunnistautuaksesi palveluun (Kuva 24).



Kuva 24 Onnistunut ohjattu liitos

Onnistuneen liitoksen jälkeen projektisi tulee näkyviin osoitteeseen https://console.firebase.google.com/. Kyseinen sivusto toimii Firebase:n konsolina ja tarjoaa tarvittavat työkalut palveluiden hallintaan.

Omassa sovelluksessani riitti pelkästään ilmoituksien lähettäminen ja niiden ajastaminen (Kuva 25). Ilmoituksien kohdentaminen tietyille käyttäjille olisi myös mahdollista, mutta tällöin sovelluksen käyttö vaatisi rekisteröitymisen, mikä ei ollut enää asiakkaan mieleen.



Kuva 25 Firebase hallinta ja ilmoituksien lähettäminen

8.2 Ilmoituksen vastaanottaminen ja käsittely sovelluksessa

Jotta sovellukseni osaa käsitellä vastaanotettavat ilmoitukset, minun tuli määritellä AndroidManifest.xml:ään uusi palvelu (Kuva 26). Palvelun parametrissä *android:name=".FirebaseService"*, viitataan FirebaseService.java tiedostoon jossa itse käsittely tapahtuu.

```
<service android:name=".FirebaseService">
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT" />
    </intent-filter>
</service>
```

Kuva 26 Palvelun määrittely AndroidManifest.xml:ssä

Palveluun määritelty action-elementti kertoo Android-käyttöjärjestelmälle, että tämä sovellus haluaa vastaanottaa ilmoituksia. Tästä kerrotaan myös varoitusluontoisesti käyttäjälle sovelluksen asennusvaiheessa.

FireBaseService.java luokka laajentaa kirjastoa FirebaseMessagingService, joka tarjoaa erilaisia ylikirjoitettavia metodeita käyttöösi. Sovelluksessani käytän onMessageReceived() -metodia, jota kutsutaan kun ilmoitus saapuu sovellukseeni (FireBaseMessagingService 2017).

Kuvassa 27 näkyy miten käsittelen ilmoituksen onMessageReceived() -metodissa ja muunnan sen näkyvään muotoon käyttäjälle. NotificationManagerilla otetaan kiinni saapuvasta ilmoituksesta ja tämän jälkeen yksittäinen ilmoitus rakennetaan hyödyntäen Notification luokkaa asettamalla sisältöparametrit.

Ilmoituksen tulee sisältää vähintään seuraavat parametrit

- Pieni kuvake → .setSmallIcon()
- Otsikko → .setContentTitle()
- Teksti → .setContentText()

(Building a Notification 2017)

```
public class FirebaseService extends FirebaseMessagingService {
    public FirebaseService() {
    }

    @Override
    public void onMessageReceived(RemoteMessage remoteMessage) {
        super.onMessageReceived(remoteMessage);

        NotificationManager NM =(NotificationManager)getSystemService(Context.NOTIFICATION_SERVICE);
        String body = remoteMessage.getNotification().getBody();

        Notification notify = new Notification.Builder(
                getApplicationContext())
                .setContentTitle("Sammakko")
                .setContentText(body)
                .setSmallIcon(R.mipmap.ikoni)
                .setAutoCancel(true)
                .setContentIntent(PendingIntent.getActivity(this,0,new Intent(),0))
                .build();

        notify.flags |= Notification.FLAG_AUTO_CANCEL;
        NM.notify(0,notify); //Nostetaan ilmoitus
    }
}
```

Kuva 27 FirebaseService.java tiedosto jossa vastaanotettava ilmoitus käsitellään.

Lopuksi kutsutaan NotificationManager -olion (NM) metodia *notify()*, joka näyttää valmiin ilmoituksen käyttäjälle.

notify() -metodi vastaanottaa kaksi arvoa, id (int) ja itse ilmoitusolion (Notification). Ilmoituksen id:tä voidaan käyttää, jos halutaan muokata myöhemmin lähetettyä ilmoitusta. Omassa sovelluksessani tälle ei ole käyttöä, joten olen määritellyt ilmoituksen id:ksi staattisen kokonaisluvun.

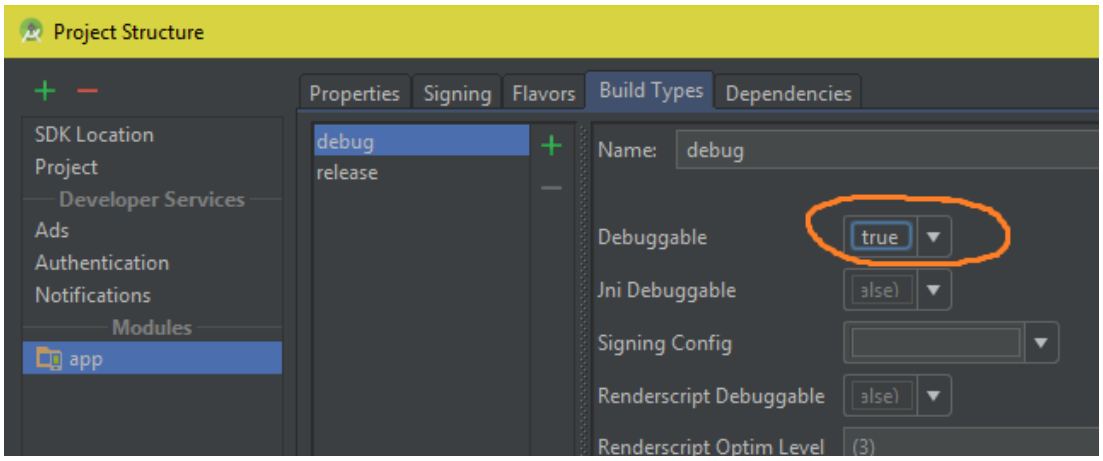
9 SOVELLUKSEN REKISTERÖINTI, KÄÄNTÄMINEN JA JULKAISU

9.1 Rekisteröityminen kehittäjäksi

Sovelluskehittäjän ja sovelluksen rekisteröityminen tapahtuu Google Play Developer Console:ssa. Sinun tulee maksaa rekisteröinnin yhteydessä kertaluontoinen maksu (25$), joka on elinikäinen. Suosittelen kuitenkin että teet asiakkaalle erillisen Google-tilin ja rekisteröit hänet kehittäjäksi, näin mahdollinen henkilökohtainen google-tunnuksesi ei sekoitu asiakkaaseen.

9.2 Allekirjoitetun APK-paketin kääntäminen

Ennen sovelluksen tuotantoversion kääntämistä, tulee virheenkorjaus kytkeä pois päältä (Kuva 28) jotta sovelluskauppa hyväksyy paketin.

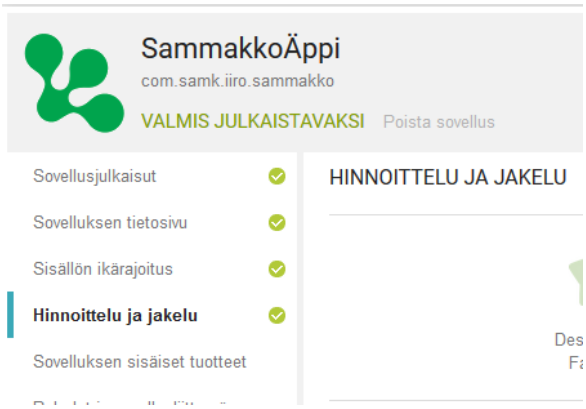


Kuva 28 virheenkorjaus tulee kytkeä pois päältä ennen paketin kääntämistä.

Aloita allekirjoitus valitsemalla Buil → Generate Signed APK… sen jälkeen aukeaa ohjattu toiminto salatun sertifikaatin luomiseksi. Valitse avaimelle tallennuspaikka ja salasana. Sertifikaatti suojaa sovelluksesi vaikka kehittäjäprofiilisi vaarantuisikin. Tämän jälkeen ohjattu toiminto luo allekirjoitetun APK-paketin, joka on valmis julkaistavaksi Google Play Storessa (Sign your app 2017).

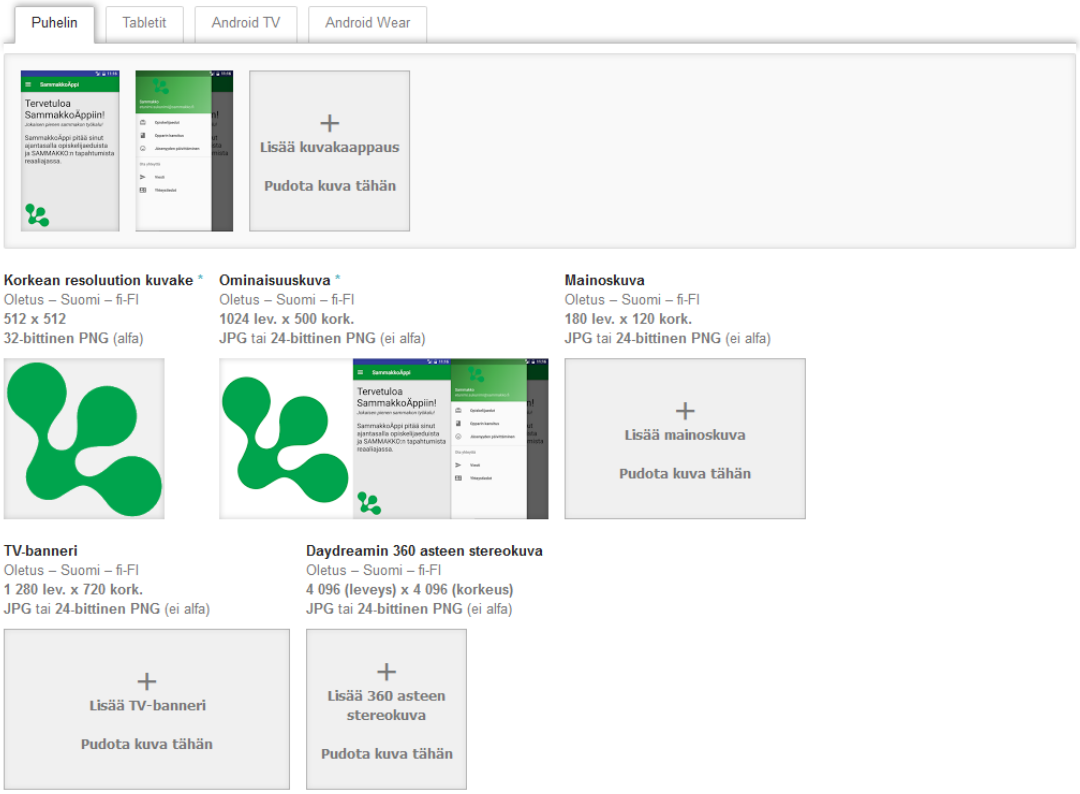
9.3 Sovelluksen julkaisu

Google Play Store vaatii sovellukseltasi monenlaisia esitietoja jotka sinun tulee täyttää. Näitä ovat mm. Sisällön ikärajoitus, Sovelluksen tietosivu, hinnoittelu ja jakelu tietoja (Kuva 29).



Kuva 29 Esitietojen täyttämistä sovelluksestasi

Kun esitiedot ovat täytetty, tulee sinun lisätä vielä tarvittavat kuvankaappaukset sovelluksestasi (Kuva 30). Nämä kuvat tulevat näkyviin sovelluskauppaan sovelluksesi ”mainos”-kuviksi.



Kuva 30 Sovelluskaupan vaatimat kuvankaappaukset

10 LOPUKSI

Sovellusten kehittäminen Androidille on tehty helpoksi. Tämä johtuu yksinomaan siitä, että sovelluksista on tullut osa Androidin sisältöä, sekä toiminnallisuutta. Google haluaa luonnollisesti tukea näitä ilmaisia sisällöntuottajia madaltamalla kynnystä kehityksen aloittamiseksi tarjoamalla ilmaiset työkalut heidän käyttöönsä.

Tulevaisuudessa uskon, että sovellukset tulevat monipuolistumaan lisääntyvien antureiden ja laitteiston tarjoamien toiminnallisuuksien myötä. Yksi hyvä esimerkki trendikkäästä tekniikasta sovelluksissa on “lisätty” todellisuus. Lisätty todellisuus tekee

tuloaan ja uskon voimakkaasti että Google haluaa kilpailla asemastaan tulla valituksi parhaana tekniikkana tai innovaationa.

Sovelluskehittäjät ovat avainasemassa keksimässä käyttötarkoituksia uusille tekniikoille.

LÄHTEET

Android.com. 2017. Viitattu 29.4.2017
https://www.android.com/

Android Studio 2017a. Meet Android Studio. 2017. Viitattu 25.2.2017
https://developer.android.com/studio/intro/index.html

Android Studio 2017b. Projects Overview. 2017. Viitattu 15.3.2017
https://developer.android.com/studio/projects/index.html

Android Studio 2017c. Application Fundamentals. 2017. Viitattu 13.3.2017
https://developer.android.com/guide/components/fundamentals.html

Android Studio 2017d. App Components. Viivattu 31.3.2017
https://developer.android.com/guide/components/fragments.html

Android Studio 2017f. Android Studio Layouts 2017f. Viivattu 31.3.2017
https://developer.android.com/guide/topics/ui/declaring-layout.html

Android developer 2017. Viitattu 27.3.2017
https://www.youtube.com/watch?v=qGoCF0Et_CU

Building a Notification 2017. Developers. Viitattu 26.4.2017
https://developer.android.com/training/notify-user/build-notification.html

Developers, AsyncTask 2017. Viitattu 25.4.2017
https://developer.android.com/reference/android/os/AsyncTask.html

Developers, Accessing Resources 2017. Viitattu 30.4.2017
https://developer.android.com/guide/topics/resources/accessing-resources.html

Developers, Exceptions 2017. Viitattu 25.4.2017
https://developer.android.com/reference/android/os/NetworkOnMainThreadException.html

FireBaseMessagingService 2017. Firebase documentation. Viitattu 26.4.2017
https://firebase.google.com/docs/reference/android/com/google/firebase/messaging/FirebaseMessagingService

JetBrains, We are JetBrains 2017. Viitattu 29.4.2017
https://www.jetbrains.com/company/?fromMenu

Square inc, Picasso 2017. Viitattu 30.4.2017
http://square.github.io/picasso/

Stackoverflow.com HelmiB 2012. Viitattu 25.4.2017
http://stackoverflow.com/questions/12575068/how-to-get-the-result-of-onpostexecute-to-main-activity-because-asynctask-is-a

Sign your app 2017. Viitattu 26.3.2017
https://developer.android.com/studio/publish/app-signing.html

Tajnar. 2017. Viitattu 26.3.2017
http://code.hootsuite.com/orientation-changes-on-android/

LIITE 1 edutFragment.java

```
package com.samk.iiro.sammakko;

import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;

import android.support.annotation.Nullable;
import android.support.v4.app.ListFragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ListView;

import java.util.ArrayList;
import java.util.concurrent.ExecutionException;


import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;


/**
 * Created by Iiro on 22.2.2017.
 */

public class edutFragment extends ListFragment implements AsyncResponse{
    LoadRssFeed asyncTask = new LoadRssFeed();

    ArrayList ReadyList = new ArrayList();

    @Override
    public void processFinish(ArrayList output) {
        ReadyList.addAll(output);
    }

    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstan-
ceState){
        View view = inflater.inflate(R.layout.edut_layout,container,false);



        try {
            ReadyList = asyncTask.get();
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (ExecutionException e) {
            e.printStackTrace();
        }


        etuAdapter etuAdapter = new etuAdapter(getContext(),ReadyList);
        ListView RSSList = (ListView) view.findViewById(android.R.id.list);

        RSSList.setAdapter(etuAdapter);
```

```
        return view;

    }

    @Override
    public void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);

        asyncTask.delegate = this;
        asyncTask.execute();
    }

    @Override
    public void onViewCreated(View view, Bundle savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);


    }

    @Override
    public void onListItemClick(ListView l, View v, int position, long id) {

        rssEtu etu = (rssEtu) l.getAdapter().getItem(position);
        String link = etu.geteLink();

        Intent browserIntent = new Intent(Intent.ACTION_VIEW, Uri.parse(link));
        startActivity(browserIntent);
        System.out.println("Stop");
    }

}
```

LIITE 2 LoadRssFeed.java

```
package com.samk.iiro.sammakko;

import android.os.AsyncTask;

import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.InputSource;

import java.net.MalformedURLException;
import java.net.URL;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

/**
 * Created by Iiro on 14.3.2017.
 */



public class LoadRssFeed extends AsyncTask<Object, Object, ArrayList<rssEtu>> {
    public AsyncResponse delegate = null;

    ArrayList<rssEtu> ListOfEtus = new ArrayList<rssEtu>();
```

```
@Override
protected ArrayList<rssEtu> doInBackground(Object... params) {
    URL url = null;
    try {
        url = new URL("https://www.feedcowboy.com/custom/fc.rss.slice.php");
    } catch (MalformedURLException e) {
        e.printStackTrace();
    }
    try {
        DocumentBuilderFactory DBF = DocumentBuilderFactory.newInstance();
        DBF.setIgnoringElementContentWhitespace(true);
        DocumentBuilder builder = DBF.newDocumentBuilder();



        Document doc;
        doc = builder.parse(new InputSource(url.openStream()));
        doc.getDocumentElement().normalize();

        NodeList list = doc.getElementsByTagName("item");
        NdLst noWhites = new NdLst(list);

        int length = noWhites.getLength();
        for (int i = 0; i < length; i++) {
            Node currentNode = noWhites.item(i);

            rssEtu item = new rssEtu();
            NodeList nodeChild = currentNode.getChildNodes();

            int cLength = nodeChild.getLength();

            for (int j = 0; j < cLength; j++) {
                String nodeName = nodeChild.item(j).getNodeName(), nodeString = null;

                if (nodeChild.item(j).getFirstChild() != null) {
                    nodeString = nodeChild.item(j).getFirstChild().getTextContent();
                }

                if (nodeString != null) {
                    if ("title".equals(nodeName)) {
                        item.seteTitle(nodeString);
                    } else if ("description".equals(nodeName)) {
                        item.seteDescription(nodeString);
                    } else if ("pubDate".equals(nodeName)) {
                        item.setePubDate(nodeString.replace(" +0200", ""));
                    } else if ("link".equals(nodeName)) {
                        item.seteLink(nodeString);
                    } else if ("image".equals(nodeName)) {
                        item.seteThumbnail(nodeString);
                    }
                }

            }
            ListOfEtus.add(item);
        }
    }catch (Exception e){
        e.printStackTrace();
    }
    return ListOfEtus;
}
```

```
    @Override
    protected void onPostExecute(ArrayList<rssEtu> result) {
        delegate.processFinish(result);
    }

}




class NdLst implements NodeList, Iterable<Node> {

    private List<Node> nodes;

    public NdLst(NodeList list) {
        nodes = new ArrayList<Node>();
        for (int i = 0; i < list.getLength(); i++) {
            Node CN = ParseWhiteSpaces(list.item(i));
            nodes.add(CN);

        }
    }

    @Override
    public Node item(int index) {
        return nodes.get(index);
    }

    @Override
    public int getLength() {
        return nodes.size();
    }

    public static Node ParseWhiteSpaces(Node n) {
        NodeList dirtNodes = (NodeList)n.getChildNodes();

        for (int i = 0; i < dirtNodes.getLength(); i++) {
            Node CN = (Node)dirtNodes.item(i);

                if(CN.getTextContent().contains("\n\t\t\t\t") ||
CN.getTextContent().contains("\n\t\t\t")){
                    n.removeChild(CN);
                } else { }
        }

        return n;
    }

    @Override
    public Iterator<Node> iterator() {
        return nodes.iterator();
    }
}
```